Parallel Omega Network Hash Construction

Chai Wen Chuah, Nurul Azma Abdullah
University Tun Hussein Onn Malaysia, Batu Pahat, 86400, Johor, Malaysia.
cwchuah@uthm.edu.my

***Abstract*—Cryptography hash function is important to ensure data integrity when the data is transmitting in the insecure connections. Merkle-Damgård construction is the well-known architecture for most hash function algorithm. This construction will take arbitrary length of input and generate a fixed length of output which best known as hash value. The process of producing the hash value is executing sequentially. The implication of this is the computation time will increase linearly when the size of input increase. Therefore, an alternative architecture that can reduce the computation time when input size is increase is needed especially in the today world where multi-core processors and multithreading programming are common. Hence, in this research an alternative Parallel Omega Network Hash Construction that can execute in multi-core machine is proposed.**

***Index Terms*—About Hash Function; Merkle-Damgård construction; Multithreading; Parallel Omega Network.**

I. INTRODUCTION

With the rapid growth of Internet, securing the integrity and confidentiality of sensitive data over insecure channels are important. One-way hash function plays a fundamental role in protecting data integrity when the data is transmitting in the insecure connections. The data integrity is the process in ensuring the data is remaining unaltered during the transmission from creation until the reception. The basic operation of hash function is to takes a variable length of messages M as input and transform the input into a fixed length of output h referred to as a hash value or hash digest, $h = H(m)$, where H is the hash function [9]. A "good" hash function has the property that hashing the arbitrary length of input M will generate the output h that are evenly distributed and apparently look random [11]. A change of single bit of the input M will resulted the change of the output h with high probability.

Hash functions are commonly built upon the Merkle-Damgård construction (MD), such as MD-5, SHA1 and SHA2 families [11]. In MD construction, the input is divided into equal-size message blocks and passes each block sequentially to a function that processes the message block. The function returns a vector value, which is then passed back to the function for the next message block. The first block vector is pre-defined vector value; the remaining vectors are dependent on the previous function's output. The hash operation is executed sequentially as the input to the function is fully dependent on the previous function's output has resulted the increase of the runtime linearly if the input size is increase.

To date, SHA2 is widely being used in many applications such as, ensuring integrity in cryptographic protocols, structuring database entries, or identifying known files in forensic investigations [6]. The SHA2 is built using MD construction. The sequential architecture of MD is recognized as a critical factor for overall hashing performance. In the era of multithreading and multi-core technology, one may need to find an alternative to increase the performance of hash function while remaining the security provided by the existing MD construction. An Omega Network Hash Construction (ONHC) proposed in [2] can execute parallel and has better performance compare with MD construction. The proposed design also provides better security in term of randomness compare with traditional MD construction [2]. However, there are some constraints in this design such as waiting time and serial time exist in the design. Hence, in this research a Parallel Omega Network Hash Construction (PON) is proposed as an alternative improvement in term of execution time compare with the existing Omega Network Hash Construction. However, the security of the proposed construction will remain same as the existing ONHC.

II. RELATED WORK

ONHC is proposed to improve the performance in term of execution time compare with the existing Merkle-Damgård construction (MD)[2]. The ONHC design is based on omega network [4] which allows hashing to be performed parallel in multi-core processors machine. SHA512 is used as the algorithm to perform the hashing. The paper shows that ONHC hashing the message faster than MD construction. The limitation for this proposal is that 'waiting time'. For example, to start executing the block function's column, it must wait for the previous block function's column execute completed. This is because the second block function's column is depending to the output from previous block function's column. Next, the security result which was carried out to examine the randomness had shown that result ONHC is better than MD construction.

Chun *et al.* [3] proposed Randomize-then-Combine Constructed Hash Function that can execute parallel multi-threaded programming paradigms. Summation and concatenation are the main functions of randomization whom claim the proposed algorithm can prevent multi-collision. An experiment to evaluate the performance in term of execution time for the proposed method is conducted in multi-threaded program. However, the result is just slightly better than or roughly same as SHA1 [3].

A methodology for generic parallelizing cryptography for hashing schemes is proposed by Atighehchi *et al.* [1]. This proposal aims to obtain optimal performances when dealing with applications that consists of multi-core target processors.

The design is based on tree hashing scheme with lower level node priority. Based on this design, there is a need to synchronize or communication between worker threads and main thread which provides the input date. Beside that's, synchronizations between threads are (almost) not required. From this, the authors claimed that by theory this strategy design may offer best performance. However, the design is just theoretical based which not yet being implemented to see the performance when execute it parallel in multi-core processor machine.

Li *et al*. [7] has proposed the parallel computation in one-way hash function. This parallel design model is based on pipeline technology. The design is hardware based design. The authors claimed that pipeline technology is an effective way to improve the performance of algorithm. The proposed design is a universal design which allow to perform under multiple instruction architecture and VLSI architecture. However, security of the hash function from the proposed design is not provided.

III. MATH AND EQUATION

Three different size of Parallel Omega Network Construction are designed (Figure 1), namely Parallel Omega Network size of 8, 16 and 32 respectively. These three different size serve as prototypes to determine the optimum size that may give the better performance when hashing three different size of message. SHA-512 algorithm is used as the function to the Parallel Omega Network Hash construction. The hashing process is simulated on dual-core, quad-core and eight-core processors machines.

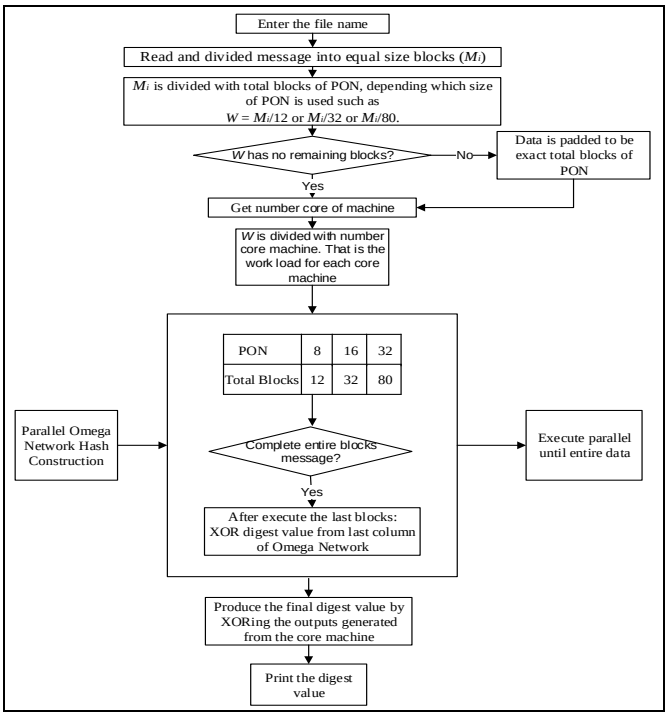

Figure 1: Parallel Omega Network Hash Construction

For better view, Figure 2 shows the process of PON 8 execute in a single thread. The message is divided into small blocks. Each block is 1024 bits. Noted that the number block of message must be multiply with total blocks on PON such as 12, 32 or 80 (Table 1). If the number of total block, the message is padding with '1000…'. In the design of the PON, each block of function takes two different size on inputs: 512 bits and 1024 bits (message block). The output from each block of function is 512 bits. The blocks on the left column initial vector 512 bits. The total number 512 bits block of pre-defined initial vector depending on the number of threads are used (Table 1). For a single thread, 4 blocks of pre-defined initial vector are needed. The pre-defined initial vector is taken from part of the square root of 2. The following columns takes the input vector from the XORed of two blocks of intermediate hash digests from previous column. For example, the Omega Network Hash Construction 8 (Figure 2), the second column function the F_5 will take the XORed intermediate hash digests from F_0 and F_2. The process proceeds for entire message. Finally, to get the final hash value for a hash message, the output of final column function blocks is XORed.

To execute the PON 8 in multithreading, the total number block of message must be multiplying by 12. The slightly different part the work load for multi-core processor will be slightly different. For example, if the simulation is carried out at the quad-core processor, the total number block of message is 121 blocks; two processors are required to execute 36 blocks each and the remaining two processors only required to execute 24 blocks of message. Lastly, output of final column function blocks for each thread is XORed to generate the hash value. This XORed operation is executed sequentially. One limitation is the size of PON increase, this final XORed process will be slower.

Overall, the process of the different size of Parallel Omega Network Hash Construction is similar. The different is the number block functions for each column (Table 1). Besides that, the number blocks of message to be executed in each thread will be different for different size of Parallel Omega Network Hash Construction.

Table 1
Set of pre-defined initial vector, set of constant value and number blocks of function for Parallel Omega Network Hash Construction.

Size of PON	8	16	32
Number block functions per column	4	8	16
Number of column	3	4	5
Total number blocks	12 multiply	32 multiply	80 multiply
Set of pre-defined initial vector	4 x p	8 x p	16 x p
Set of constant value	12 x p	32 x p	80 x p

Remark: p is the number of threads. If dual-core, then it is two threads. If quad-core, then it is four threads. If eight-core, then it is eight threads.

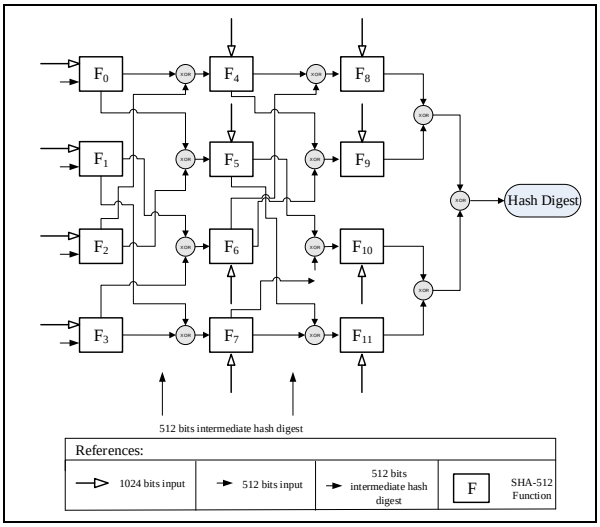

Figure 2: Omega Network Hash Construction 8

IV. Simulation And Performance Evaluation

All three designs are simulated in dual core, quad-core and eight-core machines.

Test design: SHA512 algorithm is used as the function to hash the message. The source code of SHA512 is taken from Olivier Gay [10]. The construction of hashing is PON size of 8, 16, and 32. All three designs are simulated on dual-core processors, quad-core processors and eight-core processors. The specifications of the machines are as below:

- Eight-core processors: Intel Core i7-4790 processor, 8GB DDR3 RAM
- Quad-core processors: Intel Core i5-4460 processor, 4GB DDR3 RAM
- Dual-core processors: Intel Pentium G3220 processor, 2GB DDR3 RAM.

Two lines of OpenMP commands are used to execute the proposed construction parallel. There are omp_get_num_procs() and #pragma omp parallel for num_threads(number_threads).

- omp_get_num_procs(): This OpenMP command is used to get the number of processors in the machine. The number of processors indicate the number threads is created.
- #pragma omp parallel for num_threads(number_threads): This OpenMP command is used to execute the proposed construction parallel based on the threads detected from omp_get_num_procs().

Performance test: Experiments involving measuring the execution time taken to hash three different size of files are conducted to compare the performance between hashing the files using MD construction and our proposed PONs. These sizes are 200 MB, 400MB and 600MB. For all the experiments the time is recorded for each of 100 trials. The average time (mean) for each experiment is calculated.

V. Result And Discussion

The factors evaluation the performance is included overhead, speed up, efficiency and running cost. During the execution, two types of run-time are recorded: serial run-time (T_s) and parallel run-time (T_p). The overhead is occurred in communication, synchronization, computation and memory constraints [5]. Overhead is calculated by $T_o = p.\ T_p - T_s$ [8], p is number threads or number processors. Whereas, speed up is calculated by serial run-time is divided with parallel run-time, $S = T_s / T_p$. Whereas, the efficiency (E) is a measurement of the speed up that compares to the effectively the usage of each thread $E = S / p$ [8]. Finally, the running cost ($p.T_p$) is the product of parallel run-time and the number of threads [8].

Table 2
Performance analysis of PON 8 on Dual-Core Machine

Parallel Omega Network Hash Construction – 8 (2 Threads)						
Sizes (MB)	T_s (sec)	T_p(sec)	T_o	S	E	Running cost
200	12.02	6.12	0.22	1.96	0.98	12.24
400	24.04	12.19	0.33	1.97	0.99	24.37
600	36.08	18.29	0.50	1.97	0.99	36.58

Table 3
Performance analysis of PON 8 on Quad-Core Machine

Parallel Omega Network Hash Construction – 8 (4 Threads)						
Sizes (MB)	T_s (sec)	T_p(sec)	T_o	S	E	Running cost
200	11.02	2.95	0.76	3.74	0.94	11.78
400	22.02	5.84	1.35	3.77	0.94	23.38
600	33.10	8.80	2.10	3.76	0.94	35.19

Table 4
Performance analysis of PON 8 on Eight-Core Machine

Parallel Omega Network Hash Construction – 8 (8 Threads)						
Sizes (MB)	T_s (sec)	T_p(sec)	T_o	S	E	Running cost
200	9.46	2.73	12.41	3.46	0.43	21.87
400	18.69	5.40	24.49	3.46	0.43	43.18
600	28.08	8.10	36.71	3.47	0.43	64.79

Table 5
Performance analysis of PON 16 on Dual-Core Machine

Parallel Omega Network Hash Construction – 16 (2 Threads)						
Sizes (MB)	T_s (sec)	T_p(sec)	T_o	S	E	Running cost
200	12.31	6.43	0.56	1.91	0.96	12.87
400	24.63	12.40	0.18	1.99	0.99	24.81
600	36.95	18.80	0.65	1.97	0.98	37.60

Table 6
Performance analysis of PON 16 on Quad-Core Machine

Parallel Omega Network Hash Construction – 16 (4 Threads)						
Sizes (MB)	T_s (sec)	T_p(sec)	T_o	S	E	Running cost
200	11.06	3.00	0.95	3.68	0.92	12.01
400	22.11	6.02	1.95	3.68	0.92	24.06
600	33.17	9.01	2.85	3.68	0.92	36.03

Table 7
Performance analysis of PON 16 on Eight-Core Machine

Parallel Omega Network Hash Construction – 16 (8 Threads)						
Sizes (MB)	T_s (sec)	T_p(sec)	T_o	S	E	Running cost
200	9.52	2.68	11.94	3.55	0.44	21.46
400	19.04	5.34	23.71	3.56	0.45	42.75
600	28.54	8.00	35.48	3.57	0.45	64.03

Table 8
Performance analysis of PON 32 on Dual-Core Machine

Parallel Omega Network Hash Construction – 32 (2 Threads)						
Sizes (MB)	T_s (sec)	T_p(sec)	T_o	S	E	Running cost
200	11.91	6.02	0.13	1.98	0.99	12.04
400	23.82	12.05	0.28	1.98	0.99	24.11
600	36.44	18.52	0.61	1.97	0.98	37.05

Table 9
Performance analysis of PON 32 on Quad-Core Machine

Parallel Omega Network Hash Construction – 32 (4 Threads)						
Sizes (MB)	T_s (sec)	T_p(sec)	T_o	S	E	Running cost
200	11.16	2.97	0.70	3.76	0.94	11.86
400	22.18	5.95	1.60	3.73	0.93	23.78
600	33.29	8.91	2.35	3.74	0.93	35.63

Table 10
Performance analysis of PON 32 on Eight-Core Machine

Parallel Omega Network Hash Construction – 32 (8 Threads)						
Sizes (MB)	T_s (sec)	T_p(sec)	T_o	S	E	Running cost
200	9.37	2.64	11.78	3.54	0.44	21.14
400	19.08	5.26	23.02	3.63	0.45	42.10
600	28.61	7.89	34.53	3.63	0.45	63.14

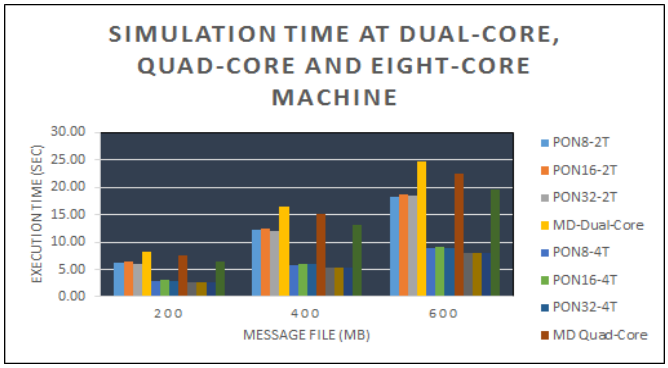

Figure 3: Execution time comparison between PONs and MD Construction, simulated on dual-core, quad-core and eight-core machine

All sizes of PON and traditional MD construction are simulated at dual-core, quad-core and eight-core machine respectively. Three sizes of message file are used to measure the performance in term of execution time for these all hash constructions. The sizes are 200MB, 400MB and 600MB. Overall, the execution time for all sizes of PON execute faster compare with the traditional MD construction in dual-core, quad-core and eight-core machines (Figure 3).

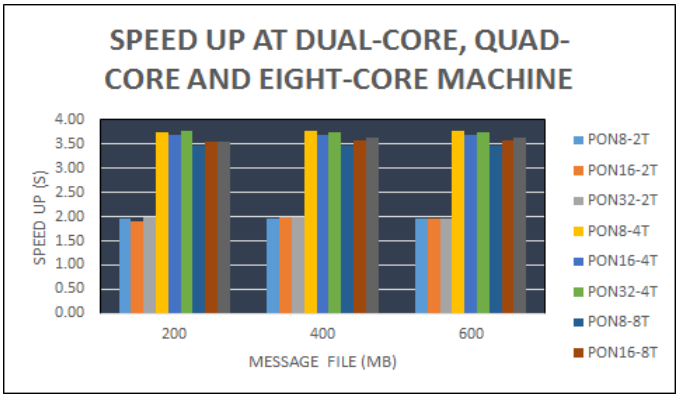

Figure 4: Speed up comparison between three sizes of PON, simulated on dual-core, quad-core and eight-core machine

PON 8, 16 and 32 are simulated at three different types of machine. The performance in term of execution time in hashing the same size of message file is similar at the same machine (Table 2-10, Figure 3). This happen because the job of hashing the message is distributed evenly to threads to execute simultaneously. The number of threads depend to the core machine. For example, dual-core machine generates two threads, quad-core machine generate four threads and eight-core machine generate eight threads. The execution time of hashing 200MB, 400MB and 600MB message files in dual-core processor, all sizes of PON take roughly 6 seconds, 12 seconds and 18 seconds respectively to generate the hash value (Table 2, Table 5, Table 8, Figure 3). In quad-core processor, all sizes of PON take approximately 3 seconds, 6 seconds and 9 seconds to hash 200MB, 400MB and 600MB message files respectively (Table 3, Table 6, Table 9, Figure 3). In eight-core processor, all sizes of PON take approximately 2.5 seconds, 5 seconds and 8 seconds to hash 200MB, 400MB and 600MB message files respectively (Table 4, Table 7, Table 10, Figure 3).

Three types of speed up calculation are presented. There are speed up based on calculation (T_s/T_p), the speed up based on Amdahl's law and the speed up based of Gustafson Barsis's law (Table 12). The fastest speed up is achieved by the PON 8 with four threads, 3.769 seconds, for input size of 400MB (Fig 4) and the average speed up is 3.75 seconds (Table 11). Based on Amdahl's law and Gustafson Barsis's law, the speed up for PON 8 still the higher one as this design only 10% of serial execution time. The lowest speed up in average it PON 32 for executing in all types of machines. This is because the serial execution time for this design is the highest consists of 30% of execution time.

Efficiency based on calculation ($(T_s/T_p)/p$), efficiency based on Amdahl's law and efficiency based of Gustafson Barsis's law are three efficiency calculation are presented. In generally, the PON 8 execute more efficiency in all machines based on three efficiency calculation. The lowest efficiency is where the designs execute at quad-core machine (Table 13, Table 14). Based on running cost calculation, the higher processors or threads, the higher cost is required to execute the proposed designs. Hence, the simulation in eight threads at eight core machine has higher running cost. The lowest running cost is simulation at dual-core machine with two threads.

Table 11
Comparison of speed up among all Parallel Omega Network Hash Construction

	2 Threads			4 Threads			8 Threads		
Sizes (MB)	8	16	32	8	16	32	8	16	32
200	1.96	1.91	1.98	3.74	3.68	3.76	3.46	3.55	3.54
400	1.97	1.99	1.98	3.77	3.68	3.73	3.46	3.56	3.63
600	1.97	1.97	1.97	3.76	3.68	3.74	3.47	3.57	3.63
Average	1.97	1.95	1.97	3.76	3.68	3.74	3.46	3.56	3.60

Table 12
Comparison of speed up among all Parallel Omega Network Hash Construction based on Amdahl's law and Gustafson Barsis's law

			Average speed up $S=T_s/T_p$			Amdahl's law speed up $S=N/[\beta N+1-\beta]$			Gustafson Barsis's law speed up $S=N-(N-1)\beta$		
PON	Serial Code (β)	Parallel Code	2T	4T	8T	2T	4T	8T	2T	4T	8T
8	0.1	0.9	1.97	3.76	3.46	1.82	3.08	4.71	1.9	3.7	7.3
16	0.2	0.8	1.95	3.68	3.56	1.67	2.50	3.33	1.8	3.4	6.6
32	0.3	0.7	1.97	3.74	3.60	1.54	2.11	2.58	1.7	3.1	5.9

Table 13
Comparison of efficiency among all Parallel Omega Network Hash Construction

Sizes	2 Threads			4 Threads			8 Threads		
(MB)	8	16	32	8	16	32	8	16	32
200	0.98	0.96	0.99	0.94	0.92	0.94	0.43	0.44	0.44
400	0.99	0.99	0.99	0.94	0.92	0.93	0.43	0.45	0.45
600	0.99	0.98	0.98	0.94	0.92	0.93	0.43	0.45	0.45
Average	0.98	0.98	0.99	0.94	0.92	0.94	0.43	0.44	0.45

Table 14
Comparison of efficiency among all Parallel Omega Network Hash Construction based on Amdahl's law and Gustafson Barsis's law

	Efficiency $S=(T_s/T_p)p$			Amdahl's law speed up $S=(N/[\beta N+1-\beta])/p$			Gustafson Barsis's law speed up $S=(N-(N-1)\beta)/p$		
PON	2T	4T	8T	2T	4T	8T	2T	4T	8T
8	0.98	0.94	0.43	0.91	0.77	0.59	0.95	0.93	0.91
16	0.98	0.92	0.44	0.83	0.63	0.42	0.90	0.85	0.83
32	0.99	0.94	0.45	0.77	0.53	0.32	0.85	0.78	0.74

VI. CONCLUSION

The main objective of designing PON is achieved as the performance in term of execution time is faster compare with MD construction. While remaining the security level in term of randomness, the proposed design has overcome the limitation in ONHC [2]. Three sizes of PON are designed (size of 8, 16, 32) and are tested on dual-core, quad-core and eight-core machines which allowed the process of hashing execute parallel which maintaining the security of hash function in term of randomness. Overall, PON 8, 16, and 32 when hash the same size of message file at the same machine take the similar execution time, speed up, overhead and efficiency. Therefore, it is hard to justify which size of PON is optimal to provide better performance. However, by considering the performance for hashing small size of message. PON 8 will execute faster as to complete one round of omega network, PON 8 just required go through 12 blocks of function compare with PON 16, 32 blocks of function and PON 32, 80 blocks of function (Table 1). The serial code of PON 8 is the lesser compare with other size of PON design (Table 12). In conclusion, PON 8 is chosen as the main design among other sizes of PONs.

ACKNOWLEDGMENT

This research was supported by STG U130, ORICC UTHM.

REFERENCES

[1] Atighehchi, K., and Muntean, T., "Generatic Parallel Cryptographic for Hashing Schemes" *IEEE 12th International Symposium on Parallel and Distributed Computing*, 2013, pp.201 – 208.

[2] Chuah, C. W., and Samsudin, A., "Omega Network Hash Construction", *Journal of Computer Sciences*, 5(12), 2009, pp.962-973.

[3] Chum, C. S., Jun, C.H., and Zhang, X. W., "Implementation of Randomize-then-Combine Constructed Hash Function", *IEEE*, 2014.

[4] Das, S., and Chaudhuri, A., "Analysis of the Effect of Size of Omega Network on its Fault Tolerance Bahaviour in Presence of Multiple Faults", *IEEE*, 1990, pp. 628-631.

[5] Grama, A., Gupta, A., Karypis, G., and Kumar, A., "Introduction to Parallel Computing", *USA: Addison Wesley Inc*, 2003.

[6] Gurjar, S., Baggili, I., Breitinger, F., and Fischer, A., "An Empirical Comparison of Widely Adopted Hash Functions in Digital Forensics: Does the Programming Language and Operating System Make a Difference?", *Proceedings of the Conference on Digital Forensics, Security and Law* , 2015, pp. 57 – 68.

[7] Li, P. Y., Shi, Y. X., and Yang, H. J., "The Parallel Computation in One-Way Hash Function Designing", *International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE)*, 2010, pp. 189 – 192.

[8] Lin, C., and Snyder, L., "Principles of Parallel Programming", *USA: Pearson Addison Wesley Inc*., 2009.

[9] Menezes, A. J., Van, O., Paul, C., and Vanstone, S. A., "Handbook of Applied Cryptography", *CRC Press*, 1996.

[10] Olivar, G., FIPS 180-2 SHA-224/256/384/512 implementation, 2007.

[11] Stallings, W., "Cryptography and Network Security Principles and Practice", *Sixth Edition. Person*, 2014.